Universidad
Europea de Madrid

LAUREATE INTERNATIONAL UNIVERSITIES

INTRODUCCIÓN A COMPILADORES Y LENGUAJES FORMALES

FUNDAMENTOS DE COMPILADORES

Índice

Presentación

El objetivo de este tema es que el estudiante comprenda la problemática asociada a los **compiladores,** cuál fue su origen, por qué son necesarios y cómo han evolucionado, además de presentar una visión global que permita identificar las distintas fases que los componen.

Los objetivos a conseguir en este tema son:

- Saber por qué son necesarios los compiladores.
- Conocer cómo nos independizamos de la máquina.
- Conocer los tipos de compiladores y las fases de un compilador.
- Reconocer cómo se diseña un compilador.
- Motivaciones.

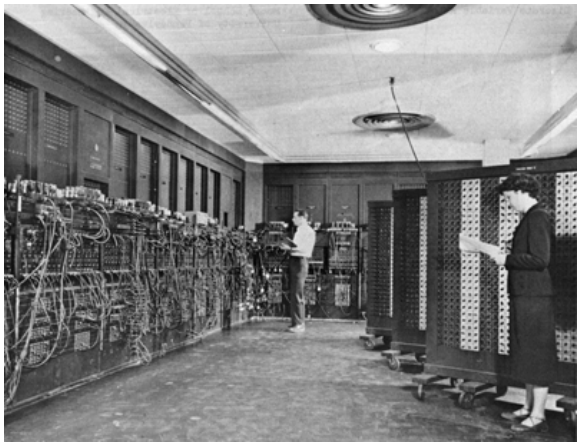
Veremos, por tanto, una reseña histórica para entender por qué era importante **independizarse de la máquina,** así como las herramientas necesarias para compilar un programa y por qué son necesarias. Debe verse el **proceso de desarrollo de un compilador** como un caso especial de ingeniería del software.

¿Por qué son necesarios los compiladores?

Con la llegada de **John Von Neumann** al proyecto **ENIAC** (1945), y tras sufrir la problemática de tener que cablear el ordenador para cada nueva tarea, decide resolver este problema. Lo consigue en 1949, escribiendo **secuencias de código o programas** que hacen que estos ordenadores realizaran los cálculos deseados. Estos programas se escribían utilizando códigos numéricos que representaban las operaciones que se iban a realizar. El lenguaje utilizado para construir estos programas se denominó **lenguaje máquina** porque estaba totalmente relacionado con el hardware de la máquina.

Como se puede ver en el siguiente ejemplo, esta forma de escribir un programa es difícil y tediosa, por lo que pronto se dio nombre a los códigos de las operaciones y a las direcciones de memoria.

Ejemplo Código máquina vs código ensamblador



Máquina ENIAC (Electronic Numerical Integrator and Computer) en Philadelphia, Pennsylvania (US Army)

Con el lenguaje ensamblador se mejoró enormemente la **velocidad y exactitud** con la que se escribían los programas. De hecho, todavía se encuentra en uso en situaciones donde se necesita velocidad y se tiene poco espacio para el código.

De todas formas, el lenguaje ensamblador no es perfecto. Sus principales **desventajas** son:

- No es fácil de escribir.
- Es difícil de leer y entender.
- Depende de la máquina para la que se ha escrito.

Ejemplo

Ejemplo de código máquina vs código ensamblador

Un ejemplo de este código máquina es el siguiente:

	Código operación	Dirección
Código máquina	**00010101**	**10000011**
Ensamblador	LOAD	X

¿Cómo nos independizamos de la máquina?

Como ya se ha indicado, el ensamblador depende totalmente de la máquina para la que se ha escrito, y esto implica **reescribir** otra vez el programa si se va a ejecutar en otra máquina. Por tanto, una vez superado el problema del cableado de la máquina para cada nuevo programa, ahora el objetivo a conseguir era independizarse de la máquina.

Es decir, se trataba de generar un código intermedio, denominado **código objeto,** que nos **ahorrara recordar las direcciones de memoria,** así como otros aspectos totalmente ligados con la máquina. Se pensaba que no iba a ser fácil, además de poco eficiente. Aquí vemos un ejemplo de lo que se estaba buscando:

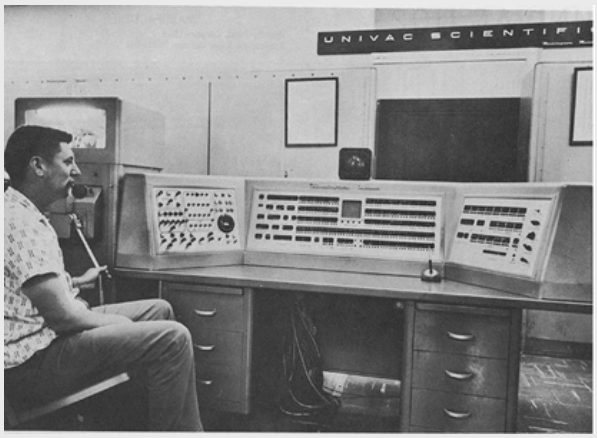
	Codigo operación	Dirección
Ensamblador	**MOV X, 5**	**10000011**
Código	X = 5	X

Había que conseguir crear un lenguaje que nos permitiera realizar acciones de tal forma que fuera **fácil de manejar y aprender** por una persona. Este tipo de lenguajes se denominó **de alto nivel,** en contraposición con los de bajo nivel (código máquina y ensamblador) totalmente dependientes de la máquina.

De esta forma, se llegó al que se denomina el **primer compilador** realizado por **Grace Hopper,** el A-0. Se empezó a trabajar en él en el otoño de 1951 y la primera rutina "compilada" se probó con éxito en un UNIVAC en la primavera de 1952 (Hopper & Mauchly, 1953). El concepto de compilador de estos momentos consistía en la aceptación del pseudocódigo, su decodificación, la búsqueda de la subrutina apropiada, la asignación de las direcciones de memoria a las subrutinas en el programa, la selección e inserción de las posiciones de memoria de los argumentos y resultados, la organización de la transferencia del control y la escritura del programa terminado en una cinta. Esto era más un cargador o enlazador que el concepto actual de compilador, pero fue la primera vez que se usó la palabra **compiler**.

Por otro lado, en 1954, John Backus comenzó el desarrollo de un compilador de **FORTRAN** para IBM, concretamente para el IBM 704, que le llevó dos años y medio y 18 hombres para realizarlo. Consistió en dos componentes: el lenguaje FORTRAN y el traductor para el IBM 704. Este desarrollo fue un **verdadero compilador** tal y como lo entendemos actualmente.

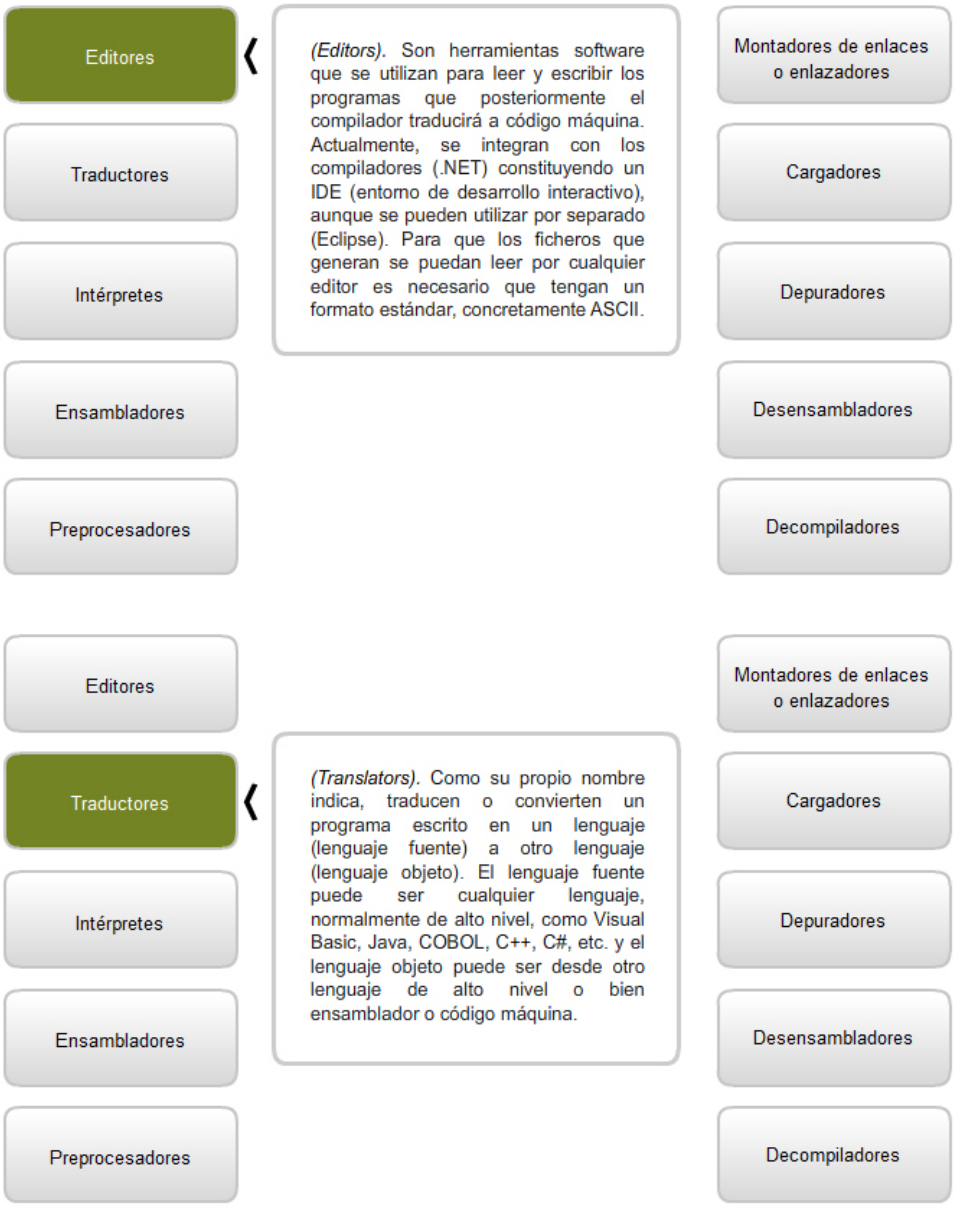
UNIVAC

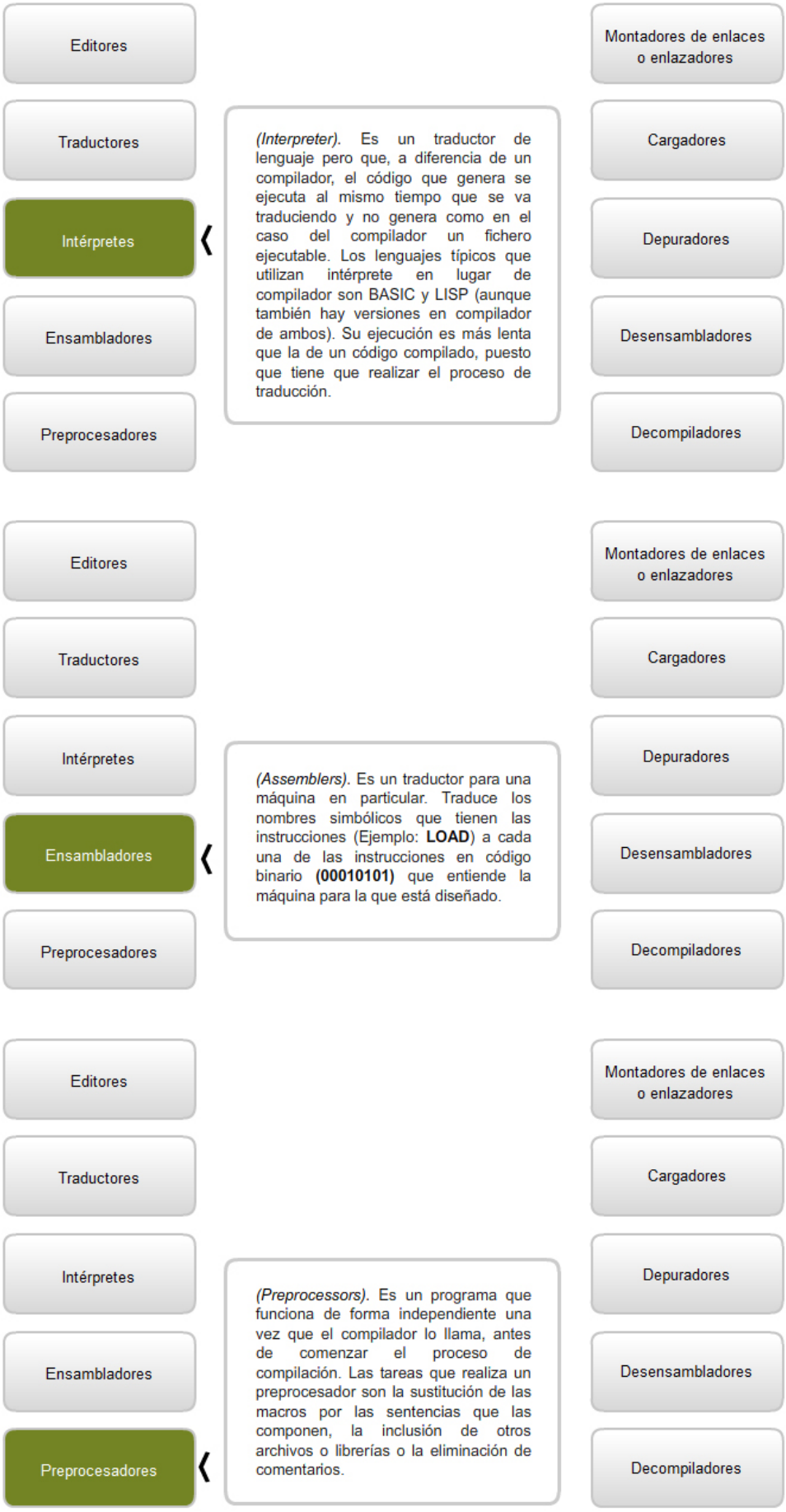


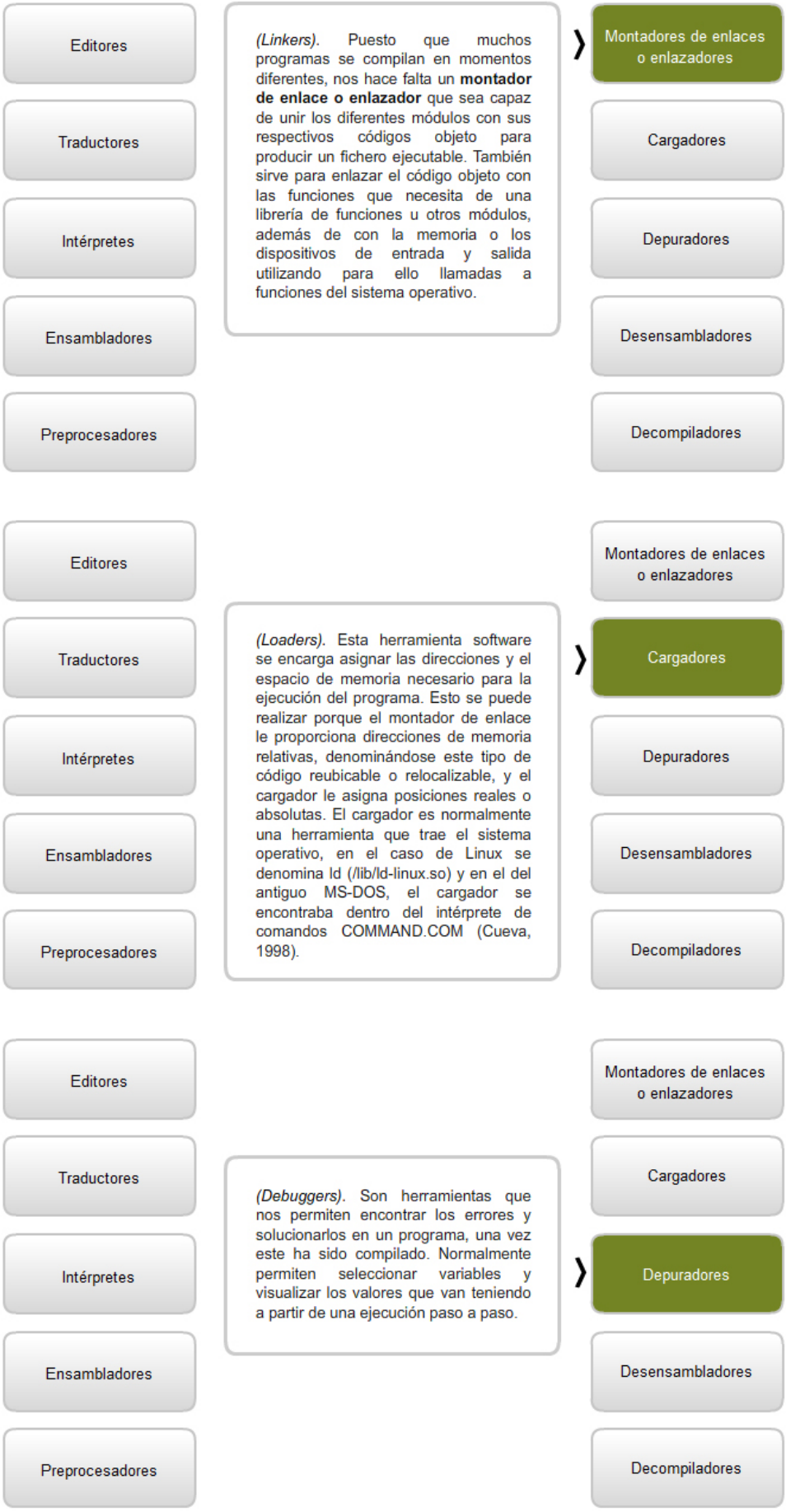
Máquina UNIVAC (US Army)

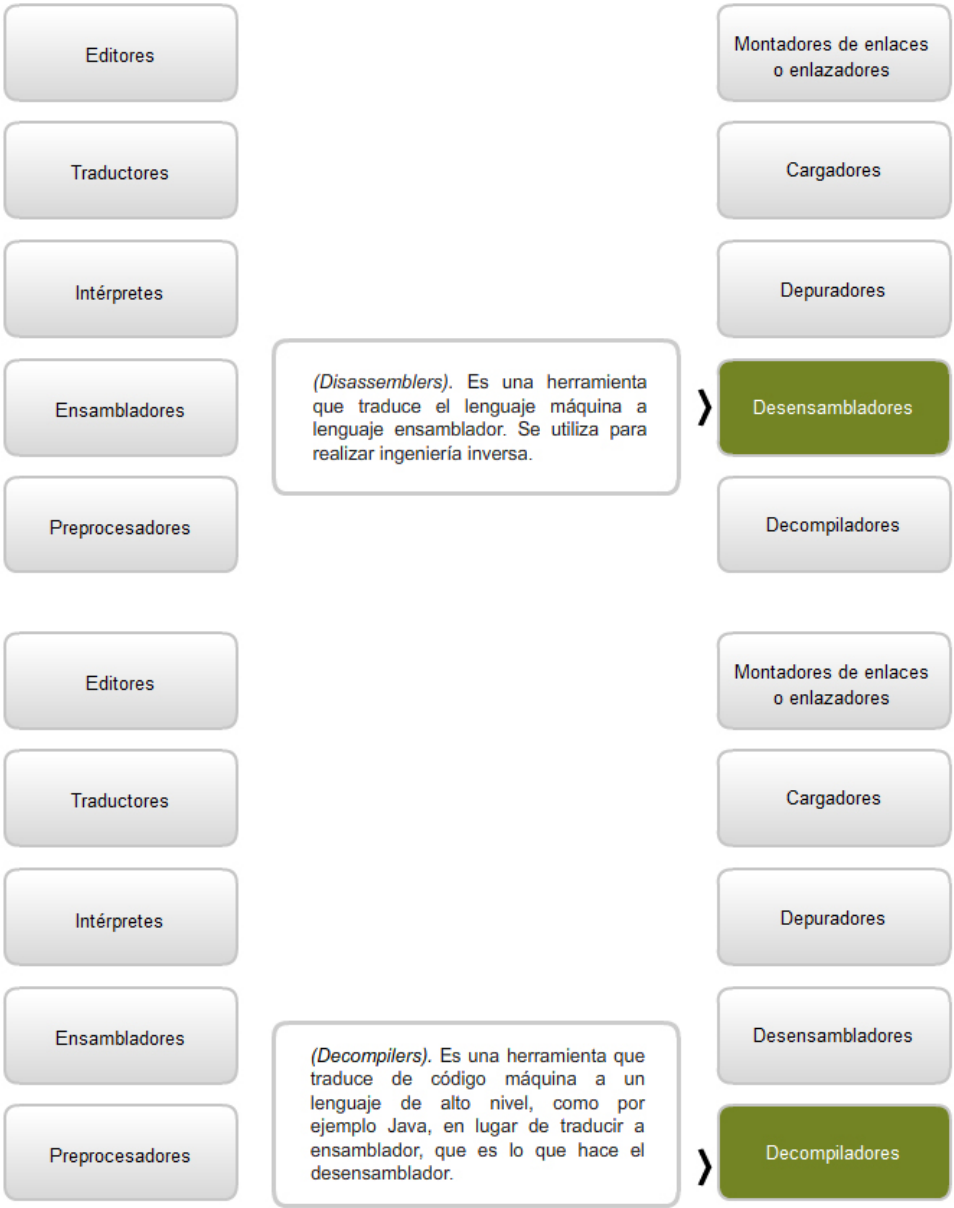
Herramientas necesarias para los compiladores

A continuación se indican las **herramientas software necesarias** para que el proceso de compilación se realice de una forma adecuada y completa:



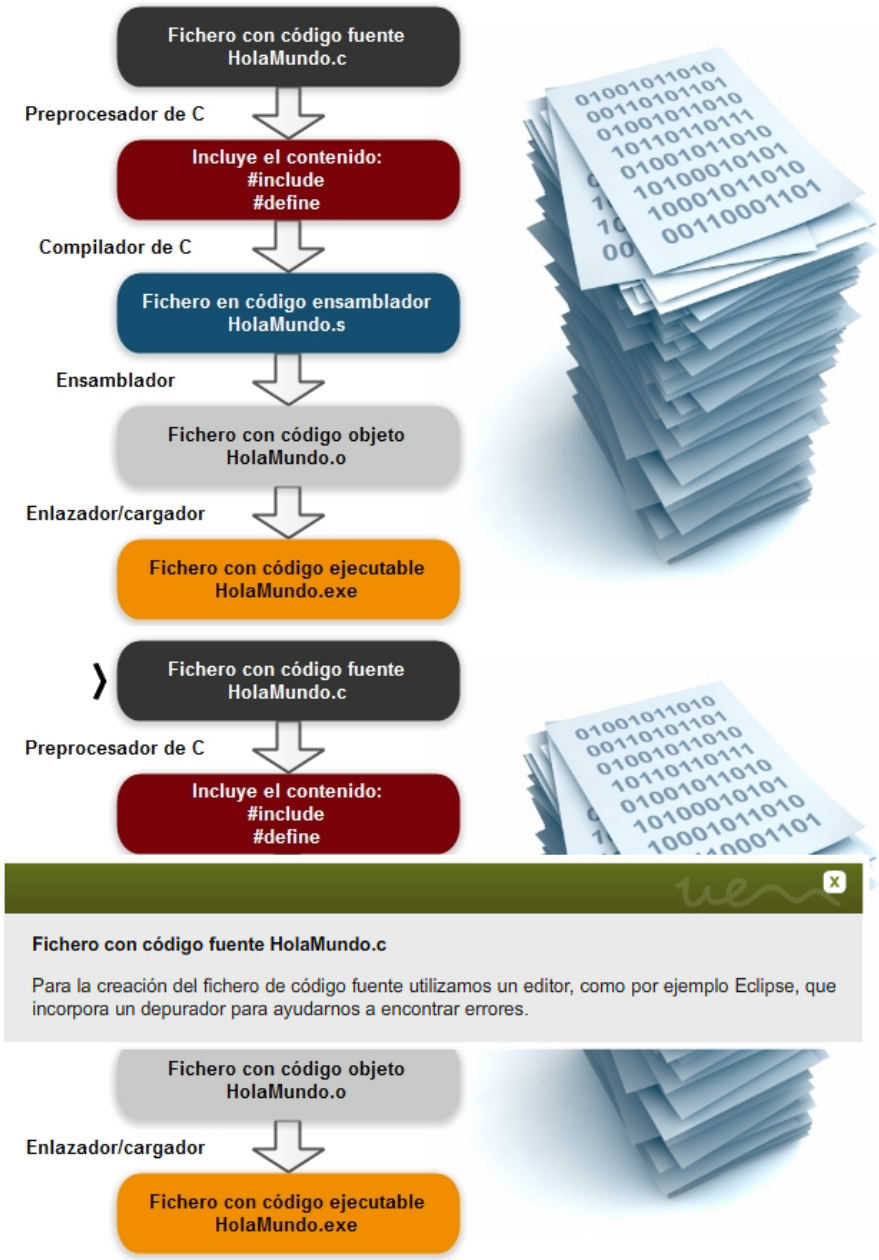


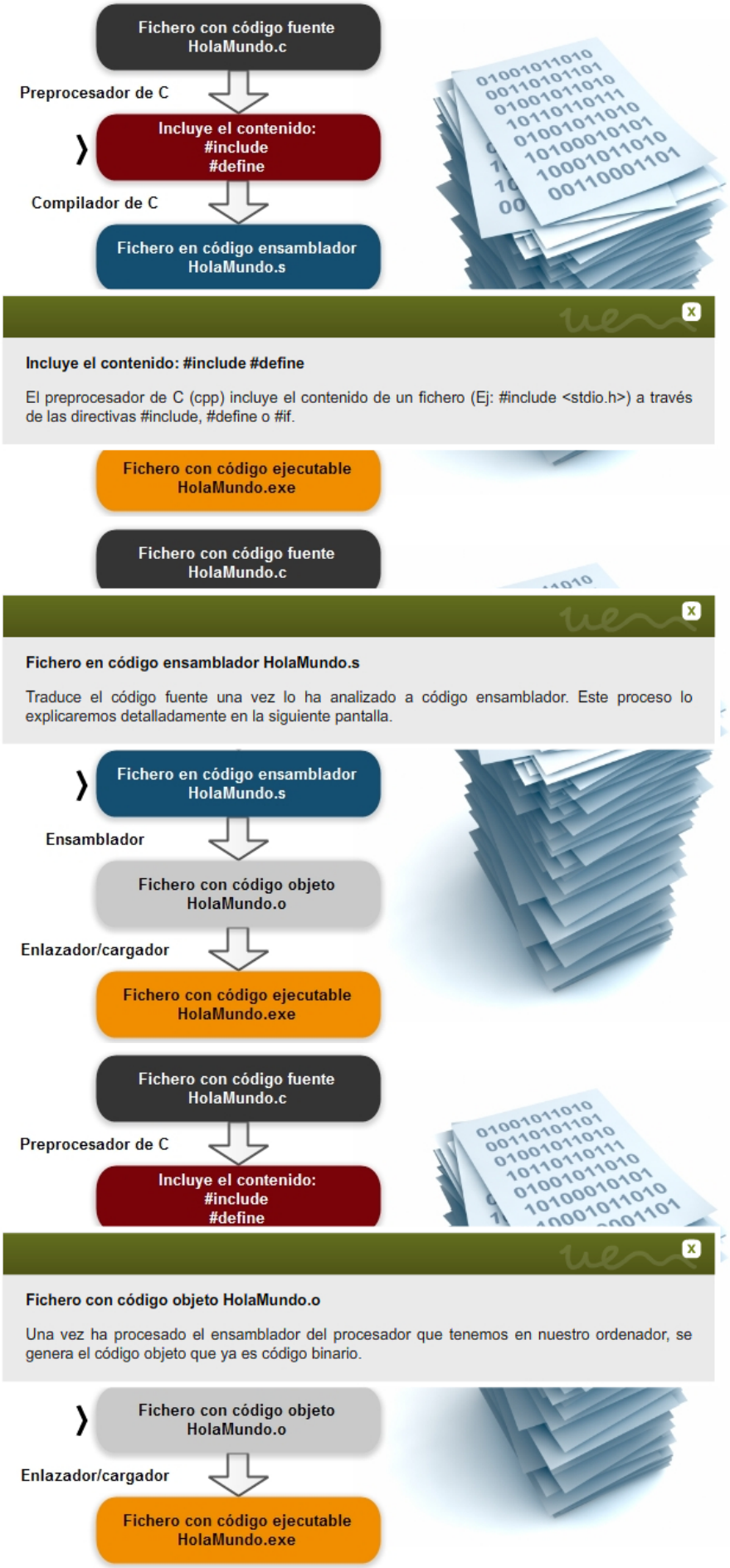


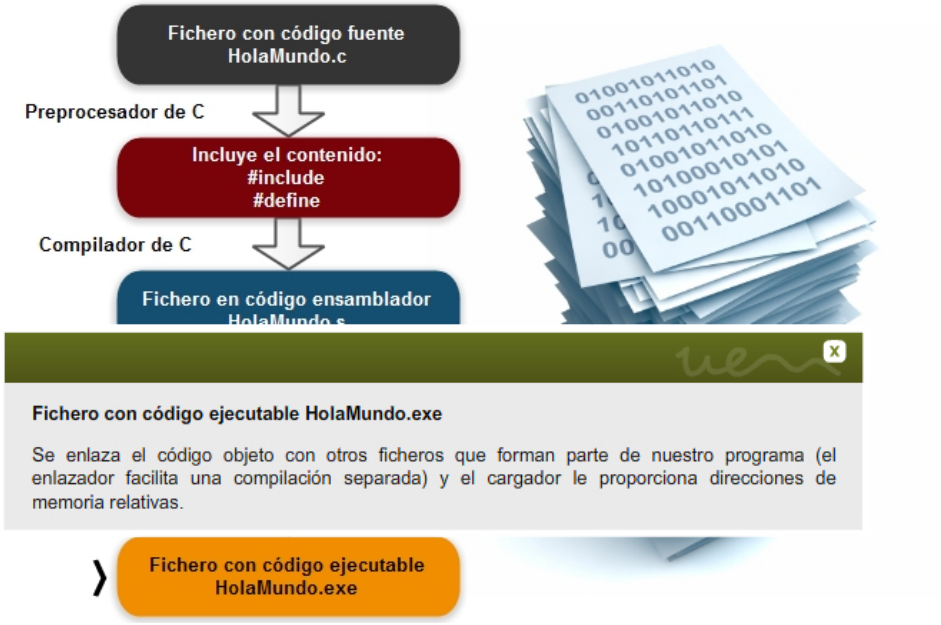


Proceso de compilación

La figura muestra el **proceso de compilación,** para, a partir de un fichero con el código fuente, obtener un fichero con el código ejecutable.

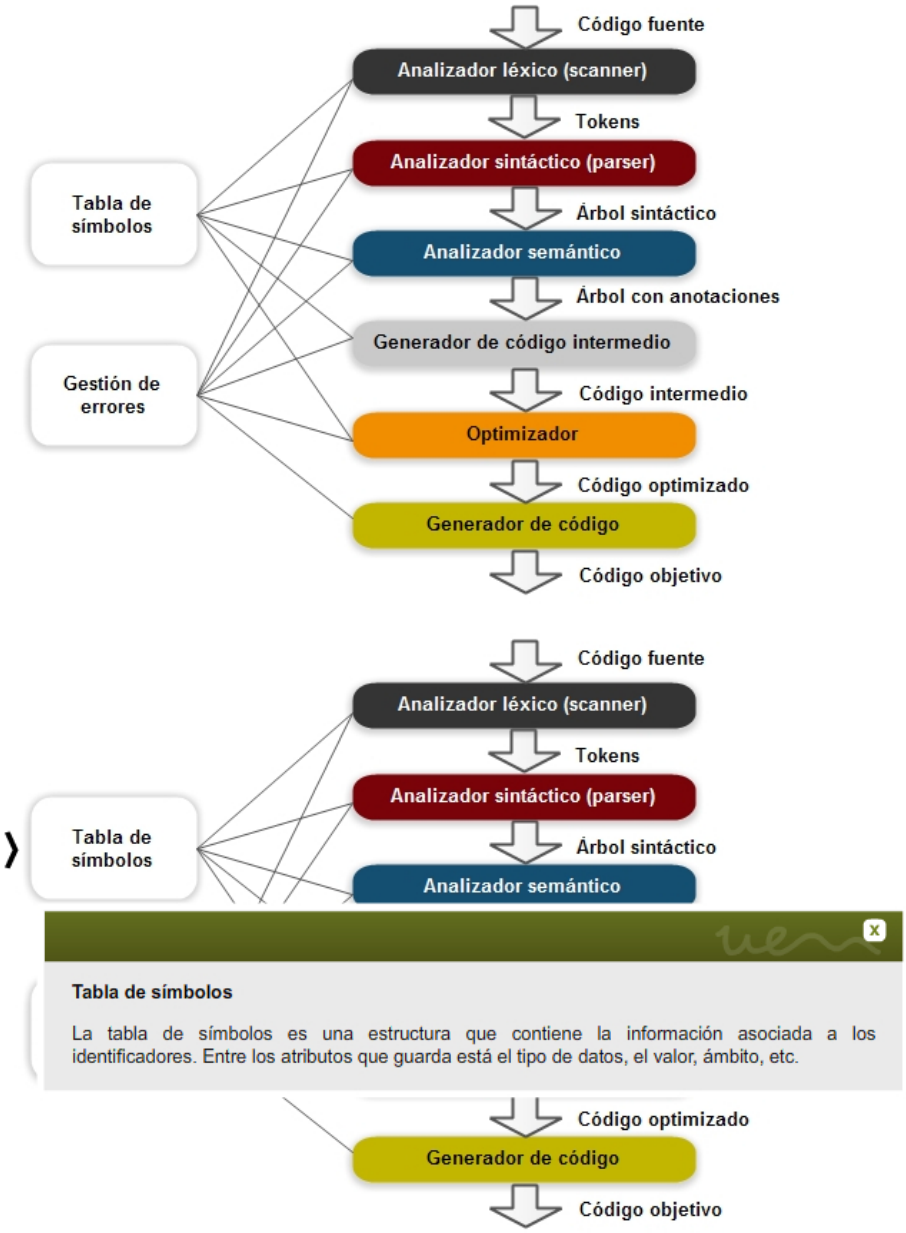


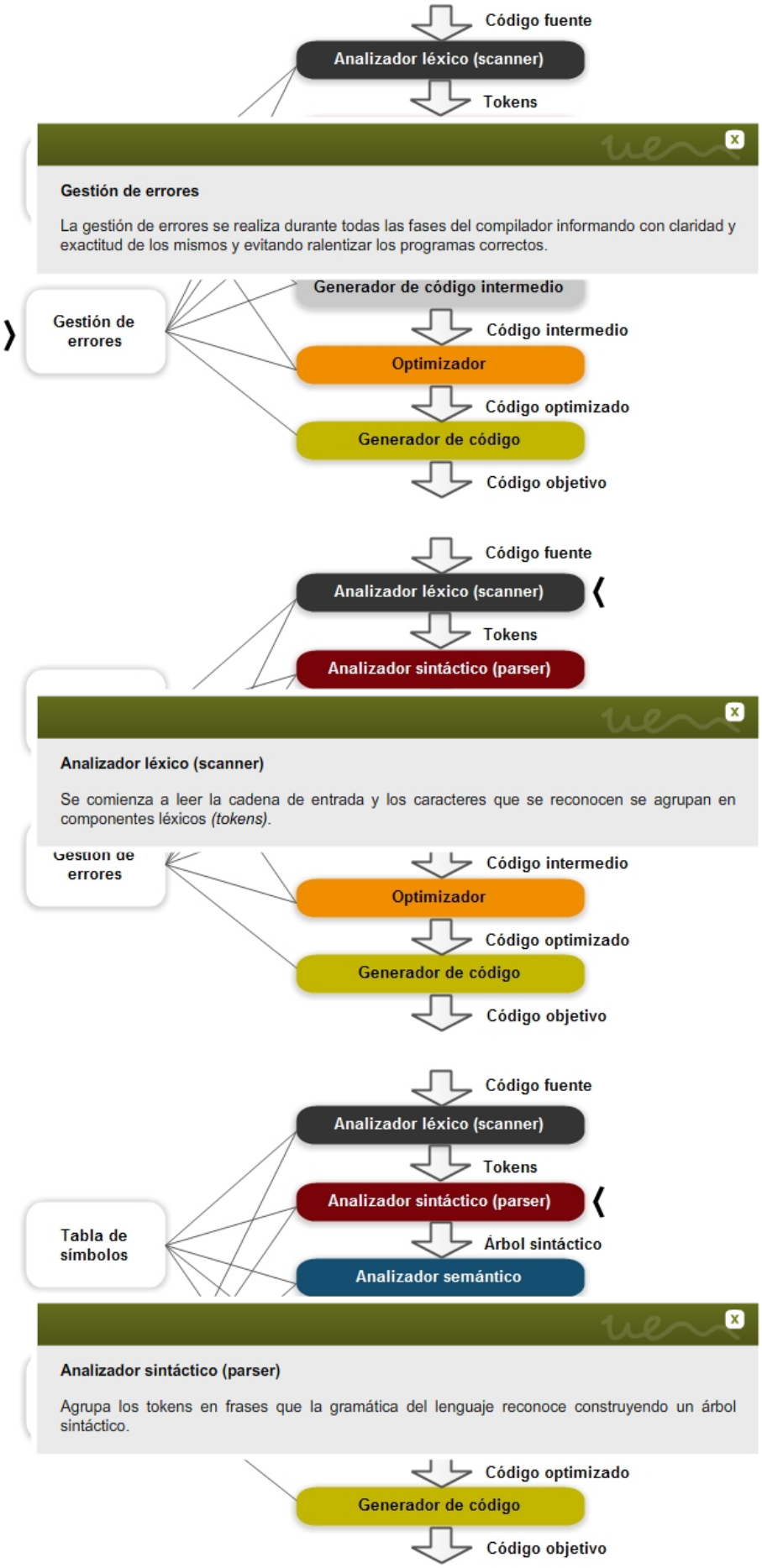


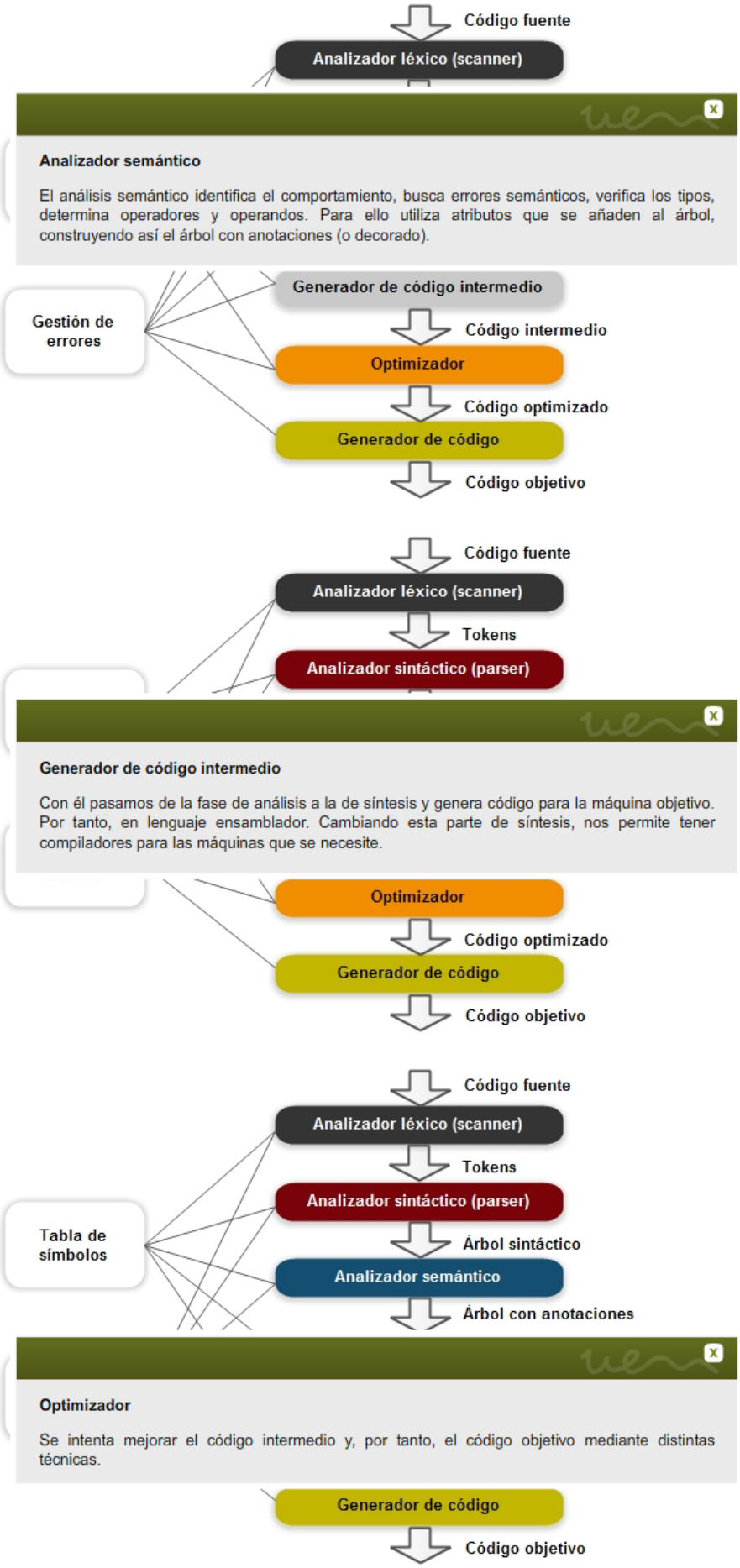


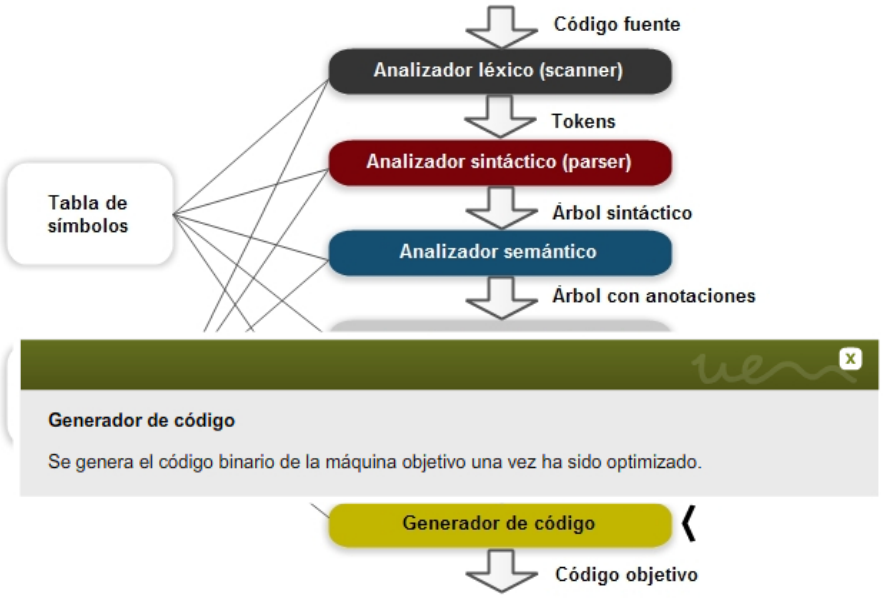
Fases de la compilación

En esta figura vemos las **fases por las que pasa un programa fuente** desde que es recibido por el analizador léxico hasta que se genera el código objeto y que va al cargador/enlazador para su ejecución.





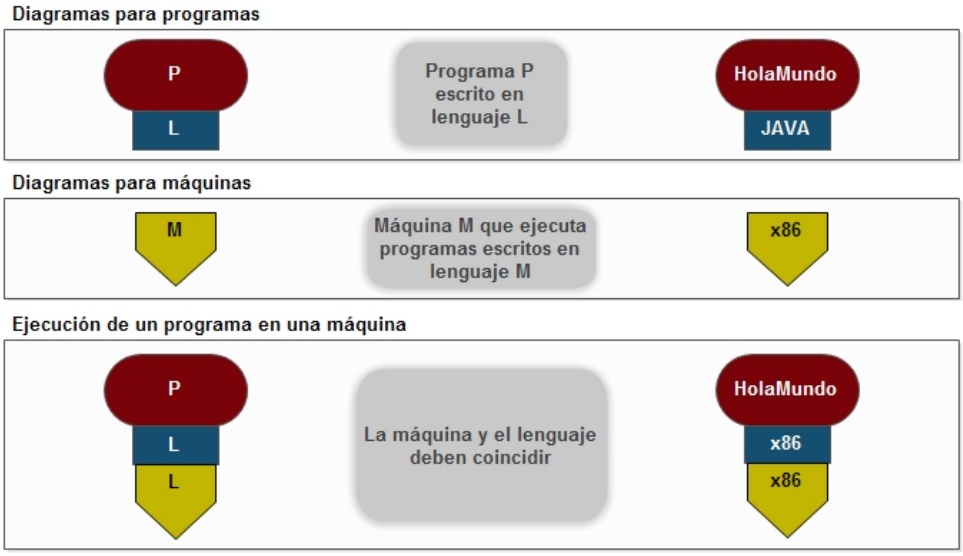




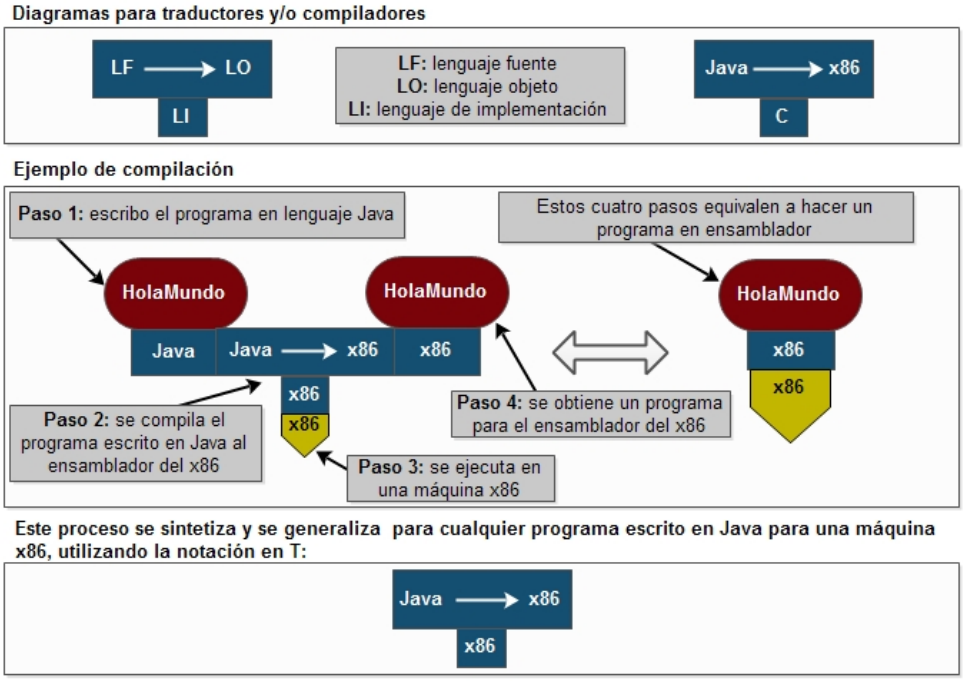
¿Cómo diseñamos un compilador?

Hasta ahora hemos partido de un lenguaje fuente y se trata de obtener el lenguaje objeto, pero no hemos tenido en cuenta el **lenguaje de implementación del propio compilador,** que es necesario para proporcionar la visión global necesaria para el diseño del mismo.

A la hora de diseñar un compilador utilizamos los diagramas de Tombstone, que son un conjunto de piezas de puzzle muy útiles para esta tarea y que a su vez tienen unas reglas:



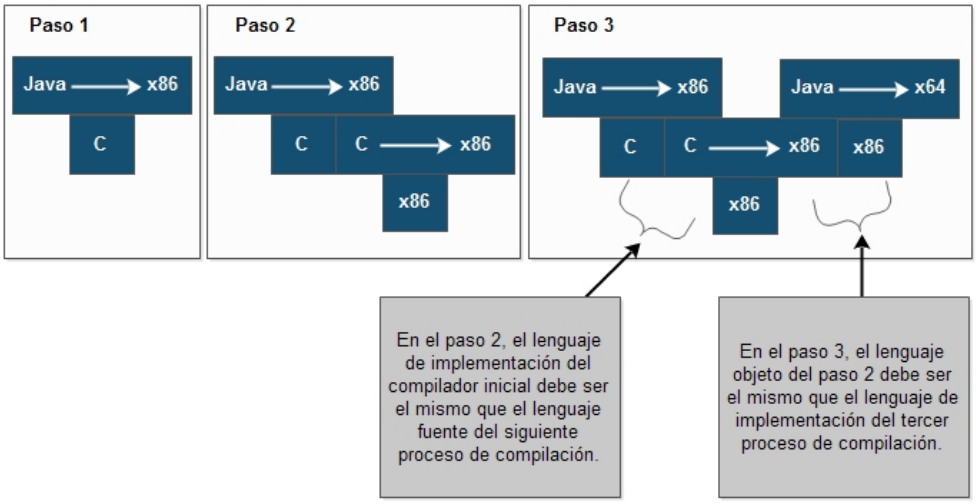
¿Cómo diseñamos un compilador?



Compilador cruzado

Los primeros compiladores se escribieron directamente para la máquina en la que se iban a ejecutar, es decir, para su código máquina. Esto, evidentemente, ya no es necesario hacerlo y si queremos que un compilador sea **portable** a otra arquitectura de máquina (por ejemplo, para cuando cambia el procesador) debemos resolver el problema: tenemos un compilador escrito en un lenguaje que queremos que se ejecute en una máquina distinta de la máquina objetivo. A esto se denomina obtener un **compilador cruzado**.

Partimos de un compilador de Java (lenguaje fuente), escrito en C (lenguaje de implementación) para una máquina x86 (lenguaje objeto), y queremos que se ejecute en una máquina de 64 bits (x64). ¿Cómo lo resolvemos?



¿Por qué es importante entender los compiladores?

Hay varios motivos que hacen que esta materia sea un componente esencial de una Ingeniería en Informática:

- Permite concentrar en una sola materia los conocimientos de Arquitectura de ordenadores, Ingeniería del software, Algoritmia, Teoría de lenguajes formales y Lenguajes de programación.
- El entender cómo se construyen los compiladores permite realizar programas más eficientes y correctos.
- Nos permite conocer mejor las decisiones de diseño que llevan a unos lenguajes a tener determinadas características: sobrecarga de operadores, polimorfismo, tipificación estática o dinámica y un largo etcétera.
- Nos permite aplicar las técnicas y herramientas a otros campos: formateadores de texto (LaTex, Tex, etc.), intérpretes gráficos (PovRAY, GIF, Postcript, etc.), lenguajes de simulación (MSDL, CBML, GPSS, etc.).
- Antes de la aparición de las técnicas que vamos a aprender, los compiladores eran muy poco eficientes.

Resumen

En este tema hemos dado un repaso a la historia de los compiladores, desde los inicios con Grace Hooper y John Von Neumann y su interés por independizarse de la máquina y pasar primero por los lenguajes de bajo nivel como ensamblador, para llegar a los de alto nivel, como FORTRAN con John Backus.

Hemos pasado por las herramientas necesarias para realizar la compilación de una forma adecuada y completa: editores, traductores, intérpretes, ensambladores, preprocesadores, enlazadores, cargadores, depuradores, desensambladores y decompiladores. Además, hemos aprendido el proceso de compilación.

Por otro lado, hemos aprendido las fases de la compilación:

- **Análisis:** léxico, sintáctico y semántico.
- **Síntesis:** generación de código intermedio, optimización y generación de código.

También hemos aprendido, por medio de los diagramas de **Tombstone,** a diseñar un compilador para otra máquina distinta de la que originalmente fue diseñado, denominándose **compilador cruzado**.

Por último, hemos comprendido por qué es necesario entender los compiladores: concentrar varias materias en una permite realizar programas más eficientes y correctos, tomar decisiones de diseño de una forma más completa y aplicar las técnicas en el desarrollo de otros tipos de herramientas relacionadas con los procesadores de lenguaje.